1 Purpose

求解 N 连环问题。若要对第 i 个环做取下或放上的操作，需满足如下两个条件之一：

(1) $i = 1$;

(2) 若 $i \neq 1$ 则需满足 $R_{i-1} = 0$ 且 $R_j = 1$, $j \leqslant i - 2$.

2 Principles

2.1 递归操作函数求解

根据上述规则，可以写出求解 N 连环问题的递归关系式。设 $R(i)$ 表示取下所有编号小于等于 i 的环；$P(i)$ 表示将所有编号小于等于 i 的环放在横梁上，则有

$$R(i) = R(i-2) + remove\,the\,ith\,ring + P(i-2) + R(i-1).$$

$$P(i) = P(i-1) + R(i-2) + put\,the\,ith\,ring + P(i-2).$$

这两个操作函数。

2.2 数据结构的抽象

为了实现 N 连环问题的求解，我们尝试将具体现实模型抽象成二进制串，用 0 和 1 表示每一位的环的状态（1 表示被取下，0 表示环在横梁上）。如此抽象之后，我们对环的取、放操作就可以用数学运算来实现了。

2.3 递归算法的 LC-3 实现

在该实验中，需要实现的递归函数为 R 和 P. 实现的关键在于不同层次之间的数据读取、返回和对递归结束条件的判断。

2.3.1 函数的返回

为了实现函数之间的数据交流，我主要用了“栈”这一数据结构。我构建了栈 A 和 B，分别用来储存函数的 PC 指针和当前层的 N 的值。

回看第一节中的操作函数，可以对某一个环的取、放操作，最多需要在第一层递归调用四次，为了保证每次递归调用结束回到本层时，程序能够寻回该层对应的 i 值，我在每次调用函数前将 i 的值压入栈 B, 在调用结束后从 B 出栈，由此保证 i 的值能够被有条不紊地传递。

除此之外，为了保证每一层函数执行结束后，函数能够成功返回上一层，我还需要在每次进入函数时，记录 PC 指针的值，由于本实验中多用 JSR 函数和 RET 函数来实现函数的调用操作，PC 的值也通常就是寄存器 $R7$ 中的值。因此我再每次进入函数时，将 $R7$ 的值压入 A 栈，在函数的出口将从 A 栈出栈的值存入 $R7$, 然后再执行 RET 指令，使程序成功跳回上一层被调用的位置。

2.3.2 递归出口的判断

由于我是用栈来存储函数跳跃指针的，因此栈 A 的栈顶指针位置就能狗反应函数是否已经运行到了最后一层的结束区。于是我在函数的结尾 RET 指令之前判断当前的栈顶指针是否指向栈底，如果指向栈底，则程序运行结束。

3 Procedure

以下是算法过程：

Step (1) 初始化：分配空间给栈 A, B; 将 N 的值读入 $R0$.

Step (2) 进入函数 R：将当前的 $R7$ 中的值压入栈 B.

Step (3) 判断 $R0$ 的值：如果为 0, 跳转至 Step (). 为 1 跳转至 Step ().

Step (4) 将 $R0$ 的值压入栈 A, 然后减去 2, 递归调用函数 R.

Step (5) 移除当前环：将 A 出栈，值存入 $R0$ 用掩码取出二进制串的当前位，然后做差，储存当前状态。

Step (6) 将 $R0$ 的值压入栈 A, 然后减去 2, 调用函数 P.

Step (7) 将 A 出栈，值存入 $R0$，将 $R0$ 的值减去 调用函数 R.

Step (8) 函数出口：判断当前栈 B 的栈顶指针是否指向栈底，如果是，则停止程序.

4 Result

输出结果满足试验要求：

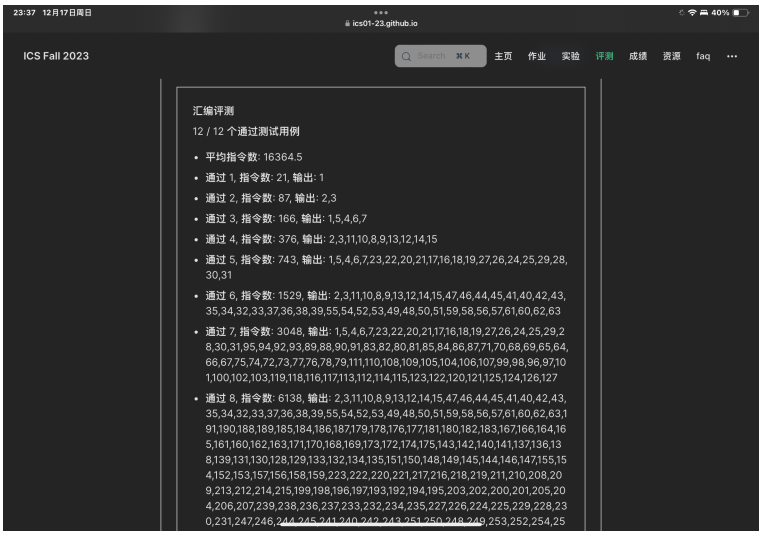

图 1: 输出结果